IBM CONFIDENTIAL (When filled in)

Prepared for IBM Attorney

LMY

*** Information Materials for IDS ***

To YOR IP Law Docket Number JP920030021 Date Sep/15/'06

Prepared by Naoko Uekusa Date of JPO Office Action Sep/05/'06

Applied Art (The following reference(s) were cited by JPO Examiner as Prior art to the following JP claims)

Ref.	Patent Document No. or Title	Publication Date (MM/DD/YY)	English abs. or counterpart document available (Y/N)	JP claim(s)
A				
B				
C				
D				
E				
F				

<NOTE> to be continued ☐

◇ PUPA : Published Unexamined Patent Application ◇ PEPA : Published Examined Patent Application

◇ PUUMA : Published Unexamined Utility Model Application ◇ PEUMA : Published Examined Utility Model Application

◇ JP : Japanese Patent ◇ * : Reference being filed before and published after the priority application date of the subject docket

Background Art (The following reference(s) were cited but not applied to the JP claims.)

Ref.	Patent Document No. or Title	Ref.	Patent Document No. or Title
bgA	PUPA 05-197565	bgD	
bgB		bgE	
bgC		bgF	

Comment, if any : to be continued ☐

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 05-197565
(43)Date of publication of application : 06.08.1993

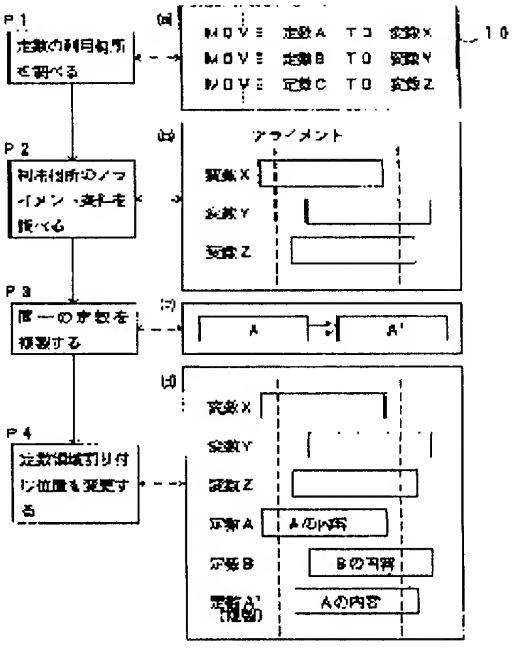
(51)Int.Cl. G06F 9/45

(21)Application number : 04-008166
(22)Date of filing : 21.01.1992

(71)Applicant : FUJITSU LTD
(72)Inventor : TAKAHASHI YOSHIO

(54) CONSTANT AREA ALLOCATION PROCESSING METHOD

(57)Abstract:
PURPOSE: To reduce the number of times referring to and updating main storage at the time of executing a processing using a constant, to speed up obtaining a translated result on a constant area allocation processing method in a compiler translating a source program into an object program.
CONSTITUTION: A place using the constant in the program is extracted by referring to a program information (P1), and an alignment condition at the place using the constant is checked (P2). When the same constant is used at plural places in the different alignment condition, the constant is copied by the number of the plural places (P3). Then, the allocation position of the constant is altered in accordance with the alignment condition of a destination using the constant (P4). Thus, optimization for adjusting the allocation areas of the respective constants to the alignment condition of a computer to be a target and reducing the number of instructions required for the reference of the respective constants is executed.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平5－197565

(43)公開日　平成5年(1993)8月6日

(51)Int.Cl.[5]	識別記号	庁内整理番号	Ｆ Ｉ	技術表示箇所
Ｇ０６Ｆ　9/45				
		9292－5B	Ｇ０６Ｆ　9/44　322　Ｈ	

審査請求　未請求　請求項の数1（全 10 頁）

(21)出願番号　特願平4－8166

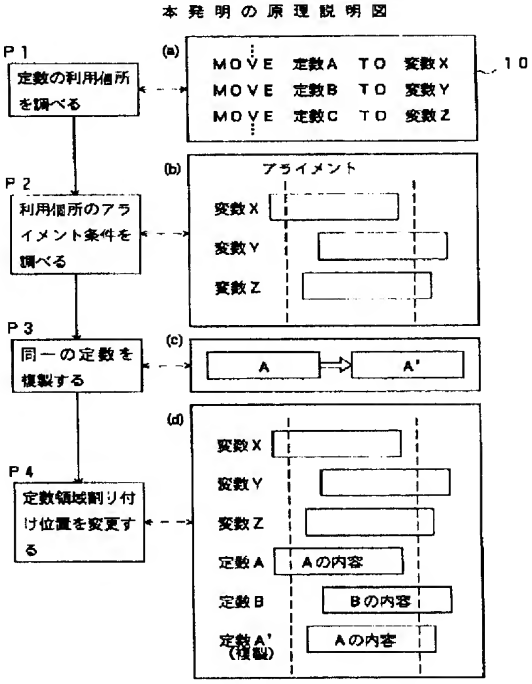
(22)出願日　平成4年(1992)1月21日

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中1015番地
(72)発明者　高橋　義雄
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内
(74)代理人　弁理士　小笠原　吉義　（外２名）

(54)【発明の名称】　定数領域割り付け処理方法

(57)【要約】
【目的】ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおける定数領域割り付け処理方法に関し，定数を利用した処理を行う際の主記憶参照および更新の回数を削減し，翻訳結果の高速化を実現することを目的とする。
【構成】プログラム情報を参照して，プログラム中の定数を利用する個所を抽出し(P1)，定数を利用する個所におけるアライメント条件を調べる(P2)。同一の定数を異なるアライメント条件のもとに複数個所で利用していれば，その定数をその複数個所分だけ複製する(P3)。そして，定数の割り付けを位置を，定数を利用する先のアライメント条件に従って変更する(P4)。こうして，各定数の割り付け領域をターゲットとなる計算機のアライメント条件に合わせ，各定数の参照に必要な命令数を少なくする最適化を行う。

本発明の原理説明図



1

【特許請求の範囲】

【請求項１】　ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおける定数領域割り付け処理方法において，プログラム情報を参照して，プログラム中の定数を利用する個所を抽出する過程(P1)と，定数を利用する個所におけるアライメント条件を調べる過程(P2)と，同一の定数を異なるアライメント条件のもとに複数個所で利用しているとき，その定数をその複数個所分だけ複製する過程(P3)と，定数の割り付け位置を，定数を利用する先のアライメント条件に従って変更する過程(P4)とを備え，各定数の割り付け領域を，ターゲットとなる計算機によって決定されるアライメント条件に合わせ，各定数の参照に必要な命令数を少なくする最適化を行うことを特徴とする定数領域割り付け処理方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は，アライメント条件に制限がある計算機で動作するプログラムのコンパイル処理方法に係り，特に，定数をその利用先のアライメントに従って割り付けることにより，定数参照の高速化を図ったコンパイラにおける定数領域割り付け処理方法に関する。

【０００２】計算機の中央処理装置（ＣＰＵ）の高速化に伴い，ＣＰＵの実行速度と主記憶への参照および更新速度との格差が問題となっている。このため，各種アーキテクチャの計算機において，効率のよい主記憶の参照と更新（以下，メモリアクセスという）方法が必要とされている。

【０００３】

【従来の技術】一般に，コンパイラは，ソースプログラムのテキストの字句解析および意味解析を行って，定数として扱うことができるデータを集め，それらの定数の領域を割り付ける処理を行っている。

【０００４】図５は従来の定数領域割り付けの例を示す図，図６は従来の命令の生成例説明図である。図５の（イ）に示すソースプログラム１０は，ＣＯＢＯＬ言語で記述されたプログラムであり，この例では，２つの文字列定数"ｆｉｒｓｔ","ｓｅｃｏｎｄ"を，変数Ｘおよび変数Ｙへ転記している。このソースプログラム１０の翻訳処理では，変数領域の先頭が８バイトアライメントにあり，図５の（ロ）に示すように，変数Ｘの領域が０から９のアドレス，変数Ｙの領域が１０から１９のアドレスに割り付けられたとする。なお，８バイトアライメントとは，アドレス値が８で割り切れる値になっていることを意味する。

【０００５】従来のコンパイラにおいては，文字列定数は，例えば図５の（ハ）に示すように，連続した領域に順番に割り付けられる。この例では，文字列定数"ｆｉｒｓｔ"が定数領域アドレスの１２８から１３２まで，文字列定数"ｓｅｃｏｎｄ"が定数領域アドレスの１３３から１３８まで割り付けられている。

2

【０００６】ところで，最近のハードウェアには，例えば２バイト単位のメモリアクセスは２バイト単位の開始アドレスからしか行えず，４バイト単位のメモリアクセスは４バイト単位の開始アドレスからしか行えないというように，メモリアクセスの単位量とメモリアクセスの開始位置とを任意に選ぶことができないアーキテクチャをとるものがある。

【０００７】このようなアライメント条件に制限があるアーキテクチャの場合には，文字列定数"ｓｅｃｏｎｄ"を転記する操作などの際に，開始アドレスが合わないために，複数バイトずつまとめて転記できないことがある。

【０００８】図６の（イ）は，図５の（イ）に示すソースプログラム１０中の２つのＭＯＶＥステートメントを翻訳した例を示している。同図では，説明を分かりやすくするために，翻訳結果をアセンブラ命令の形式で表している。ここで，各命令は次の機能を持っている。

【０００９】load-address：第１オペランドのアドレスを，第２オペランドにロードする。

load-4bytes ：第１オペランドの内容（４バイト）を，第２オペランドにロードする。

【００１０】store-4bytes：第１オペランドの内容（４バイト）を，第２オペランドの示すアドレスにストアする。

load-1byte　：第１オペランドの内容（１バイト）を，第２オペランドにロードする。

【００１１】store-1byte ：第１オペランドの内容（１バイト）を，第２オペランドの示すアドレスにストアする。

このプログラムは，まず変数Ｘのアドレスをレジスタｒ０にロードし，次に図５の（ハ）に示す定数領域の１２８番地から４バイトの文字列定数"ｆｉｒｓ"をレジスタｒ１にロードし，それをレジスタｒ０の示すアドレスにストアしている。続いて，定数領域の１３２番地から１バイトの文字列定数"ｔ"をレジスタｒ１にロードし，それをレジスタｒ０の示すアドレスから４バイト目のアドレスにストアし，以下，変数Ｙのアドレスをレジスタｒ０にロードし，同様に"ｓ","ｅ","ｃ","ｏ","ｎ","ｄ"を１バイトずつロードして，変数Ｙの領域に順番にストアしている。

【００１２】この結果，文字列定数"ｆｉｒｓｔ","ｓｅｃｏｎｄ"の変数Ｘおよび変数Ｙへの転記は，図６の（ロ）に示すように行われることになる。ここで，文字列定数"ｓｅｃｏｎｄ"が１バイトずつしか転記されないのは，定数"ｓｅｃｏｎｄ"が奇数番地から始まり，変数Ｙが偶数番地から始まるため，２バイト単位以上まとめてロード／ストアすることができないからである。

【００１３】

【発明が解決しようとする課題】以上のように，従来の

コンパイラにおいては，定数を参照するために必要な命令数に関する考慮を払うことなく，集めた定数を機械的に連続した領域に割り付けているため，転記元と転記先の開始アドレスが合わないときに，複数バイトをまとめて転記することができず，命令数が多くなって，命令の実行時間が長くなるという問題があった。

【００１４】本発明は上記問題点の解決を図り，定数を利用した処理を行う際の主記憶参照および更新を効率よく行うことができるようにし，翻訳したプログラムの高速実行を可能とすることを目的としている。

【００１５】

【課題を解決するための手段】図１は本発明の原理説明図である。ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおいて定数領域を割り付ける際に，字句・意味解析で見つけた定数を，単純に連続領域に割り付けるのではなく，それぞれの定数が利用される先のアライメント条件を調べ，翻訳結果を動作させるターゲットとなる計算機上でのメモリアクセスの効率が最大限になるように，定数割り付け領域の最適化を行う。

【００１６】そのため，処理過程Ｐ１では，字句・意味解析を行った結果のプログラム情報を参照して，プログラム中において定数を利用する個所を抽出する。処理過程Ｐ２では，処理過程Ｐ１で抽出した定数の利用個所におけるアライメント条件を調べる。すなわち，定数を転記する個所が何バイト単位から始まっているかなどを調べる。

【００１７】処理過程Ｐ３では，同一の定数を異なるアライメント条件のもとに複数個所で利用しているとき，その定数をその複数個所分だけ複製する。処理過程Ｐ４では，定数の割り付けを位置を，定数を利用する先のアライメント条件に従って変更する。例えば，定数の利用先が４バイトアライメントになっていれば，その定数についても４バイトアライメントに合わせるように割り付ける。

【００１８】これにより，各定数の割り付け領域を利用先のアライメントに合わせ，定数参照に必要な命令数を少なくする。

【００１９】

【作用】本発明は，ハードウェアアーキテクチャにより決定されるアライメント条件を守りながら，定数をその利用先のアライメントに従って割り付けることにより，定数参照の高速化を図るものである。

【００２０】処理過程Ｐ１では，例えば図１の(a) に示すようなソースプログラム１０の定数の利用個所を調べる。この例では，変数Ｘ，Ｙ，Ｚにそれぞれ定数Ａ，Ｂ，Ａを転記している。処理過程Ｐ２では，定数利用個所のアライメント条件，すなわち，変数Ｘ，Ｙ，Ｚが各々どのようなアライメントで配置されているかを調べる。処理過程Ｐ３では，同一の定数Ａを利用する個所が２箇所あるので，定数Ａの複製Ａ’を作る。処理過程Ｐ４では，定数Ａ，定数Ｂ，定数Ａ’のアライメントを，それぞれ変数Ｘ，Ｙ，Ｚのアライメントに合わせるように，各定数領域の割り付け位置を変更する。

【００２１】こうすることにより，定数Ａ，Ｂ，Ａ’を変数Ｘ，Ｙ，Ｚに転記する際に，複数バイトずつまとめて転記する命令を用いることができるようになる。

【００２２】

【実施例】図２は本発明の実施例による処理構成を示すブロック図，図３は本発明の実施例による定数領域割り付けの例を示す図，図４は本発明の実施例による命令の生成例説明図である。

【００２３】図２において，ソースプログラム１０は，ＣＯＢＯＬ言語などの高級言語で記述された翻訳対象のプログラムである。処理装置１１は，ＣＰＵおよびメモリなどからなるコンパイル処理装置である。コンパイラ１２は，ソースプログラム１０を機械語レベルのオブジェクトプログラム２２に翻訳するプログラムである。

【００２４】ソース解析部１３は，ソースプログラム１０の字句解析および意味解析を行い，解析結果を中間コードなどのプログラム情報１４として出力する。最適化処理部１５は，プログラム情報１４を参照して，処理の最適化を行うものである。本発明は，特にこの最適化処理部１５の処理に関係している。

【００２５】アーキテクチャ情報１６は，ターゲットとなる計算機のアライメント条件などに関する事前に与えられる情報である。アライメント条件判定処理部１７は，アーキテクチャ情報１６を参照し，オブジェクトプログラム２２を動作させる計算機に，アライメント条件に関する制限があるかどうか，すなわちメモリアクセスの単位量とメモリアクセスの開始位置との間に，何らかの制限があるかどうかを判定する。制限がない場合，従来と同じ定数領域の割り付けを行う。

【００２６】アライメントに関する制限が存在する場合，利用先確認処理部１８は，プログラム情報１４を参照して，定数の利用先を調べる。また，同一の定数がプログラム中の複数個所で参照されているかどうかを調べる。

【００２７】同一の定数が複数個所から参照されている場合に，領域複製処理部１９は，その定数を参照先の個数分だけ複製する。定数領域割り付け位置変更処理部２０は，定数の利用先のアライメント情報をプログラム情報１４から入手し，それに合わせて各定数の割り付け位置を変更する処理を行う。変更した位置に関する情報は，プログラム情報１４として格納しておく。

【００２８】オブジェクト生成処理部２１は，プログラム情報１４に従って，オブジェクトプログラム２２を生成する処理を行うものである。以上により，ハードウェアのアライメント条件のもとで，高速に参照可能な定数領域の割り付けが実現されることになる。

【００２９】例えば，図３の（イ）に示すようなＣＯＢ

ＯＬ言語で記述されたソースプログラム１０をコンパイルするものとする。このソースプログラム１０は，従来技術の説明で用いた例と同じである。

【００３０】変数Ｘは，図３の（ロ）に示すように，８バイトアライメントにある変数領域アドレスの０から確保され，変数Ｙは，変数領域アドレスの１０から確保されている。この変数Ｘおよび変数Ｙの確保されるアドレスは，ＴＥＳＴという名前で構造化されているため，変更することはできない。

【００３１】変数Ｘ，変数Ｙに代入する文字列定数"ｆｉｒｓｔ"，"ｓｅｃｏｎｄ"の領域を割り付ける場合，変数Ｘは，変数領域アドレスの８バイトアライメントから確保されているので，文字列定数"ｆｉｒｓｔ"も８バイトアライメントから確保する。この例では，定数領域アドレスの１２８に文字列定数"ｆｉｒｓｔ"を割り付けている。

【００３２】一方，変数Ｙは，変数領域アドレスの１０，すなわち８バイトアライメントより２バイト後の位置に確保されているので，文字列定数"ｓｅｃｏｎｄ"は８バイトアライメントより２バイト後に確保する。そのため，例えば図３の（ハ）に示すように，文字列定数"ｆｉｒｓｔ"の後に５バイトの未使用領域を設け，文字列定数"ｓｅｃｏｎｄ"を定数領域のアドレス１３８から確保する。

【００３３】図４の（イ）は，図３の（イ）に示すソースプログラム１０中の２つのＭＯＶＥステートメントを翻訳した例を示している。ターゲットの計算機は，いわゆるＲＩＳＣアーキテクチャの命令セットを採用する計算機で，メモリアクセスの単位量とメモリアクセス開始位置とを任意に選ぶことができないというアライメント条件の制限がある。図４の（イ）に示す各命令は次の機能を持っている。

【００３４】load-address：第１オペランドのアドレスを，第２オペランドにロードする。

load-1byte　：第１オペランドの内容（１バイト）を，第２オペランドにロードする。

【００３５】store-1byte　：第１オペランドの内容（１バイト）を，第２オペランドの示すアドレスにストアする。

load-2bytes　：第１オペランドの内容（２バイト）を，第２オペランドにロードする。

【００３６】store-2bytes：第１オペランドの内容（２バイト）を，第２オペランドの示すアドレスにストアする。

load-4bytes　：第１オペランドの内容（４バイト）を，第２オペランドにロードする。

【００３７】store-4bytes：第１オペランドの内容（４バイト）を，第２オペランドの示すアドレスにストアする。

このプログラムは，まず変数Ｘのアドレスをレジスタｒ０にロードし，次に図３の（ハ）に示す定数領域の１２８番地から４バイトの文字列定数"ｆｉｒｓ"をレジスタｒ１にロードし，それをレジスタｒ０の示すアドレスにストアする。続いて，定数領域の１３２番地から１バイトの文字列定数"ｔ"をレジスタｒ１にロードし，それをレジスタｒ０の示すアドレスから４バイト目のアドレスにストアする。

【００３８】次に変数Ｙのアドレスをレジスタｒ０にロードし，変数Ｙの開始位置と文字列定数"ｓｅｃｏｎｄ"の開始位置が，ともに８バイトアライメント＋２の位置にあるので，先頭の２バイト"ｓｅ"をロードし，それをレジスタｒ０の示すアドレスにストアする。残りの文字列定数"ｃｏｎｄ"は８バイトアライメント＋４，すなわち４バイトアライメントにあるので，４バイトをまとめてロードし，レジスタｒ０の内容＋２のアドレス（変数領域のアドレス１２）にストアする。

【００３９】この例では，定数の参照個所がそれぞれ１箇所であるが，もし同一内容の定数が複数個所で参照される場合には，アライメントを合わせるのに必要な分の定数の複製を作ることにより，定数領域割り付け位置のアライメントを，各利用先のアライメントに合わせる。

【００４０】このように，アライメントを合わせて領域を確保するため，複数バイトをまとめてコピーすることができる。したがって，図６に示す従来方法と比べると明らかなように，定数参照の命令数を削減し，翻訳結果の高速化が実現される。

【００４１】

【発明の効果】以上説明したように，本発明によれば，主記憶に対する参照および更新の回数を削減でき，翻訳結果の高速化が可能になる。したがって，計算機全体の性能向上に寄与するところが大きい。

【図面の簡単な説明】

【図１】本発明の原理説明図である。

【図２】本発明の実施例ブロック図である。

【図３】本発明の実施例による定数領域割り付けの例を示す図である。

【図４】本発明の実施例による命令の生成例説明図である。

【図５】従来の定数領域割り付けの例を示す図である。

【図６】従来の命令の生成例説明図である。

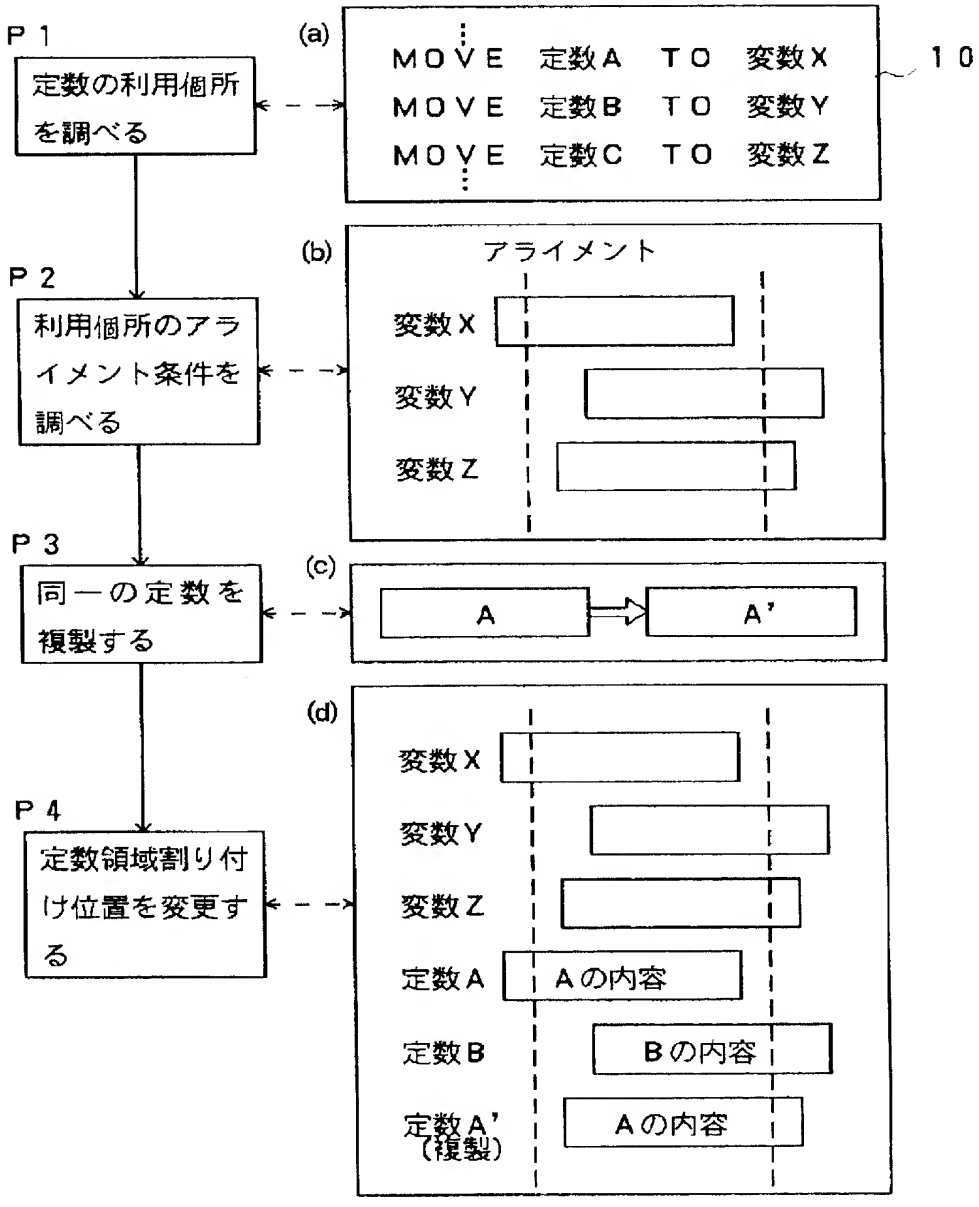
【符号の説明】

１０　　ソースプログラム

Ｐ１～Ｐ４　　本発明の処理過程

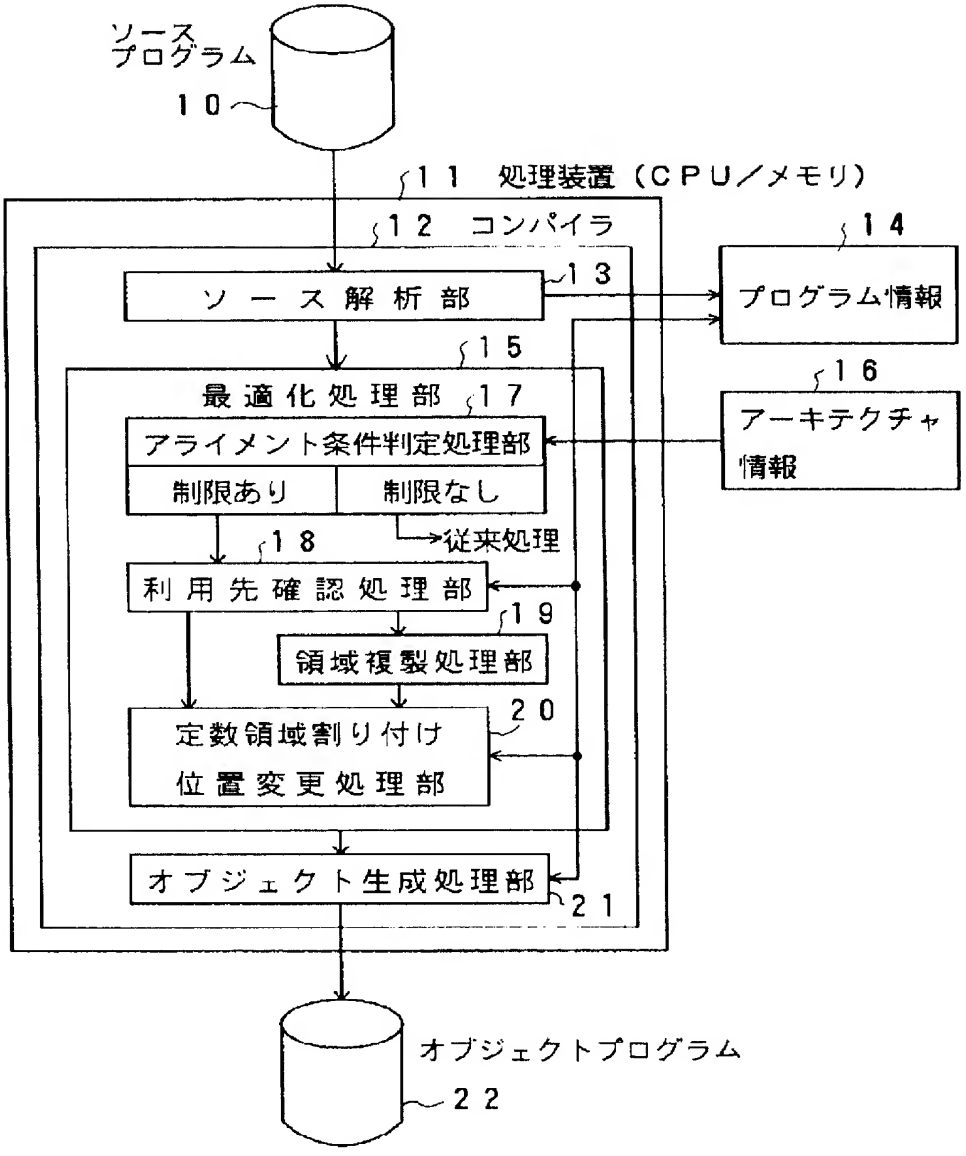
【図1】

本発明の原理説明図



【図２】

本発明の実施例ブロック図



【図3】

定数領域割り付けの例

(イ)　ソースプログラム

```
01  TEST.
    02  X  PIC  X(10).      ～10
    02  Y  PIC  X(10).

...
MOVE  "FIRST"  TO  X.
MOVE  "SECOND"  TO  Y.
```

(ロ) 変数領域

0 1 2 ・・・ 9	10 11・・・ 19
X の領域	Y の領域

(ハ) 定数領域

定数領域アドレス	16進表現	ASCII表現	8バイトアライメントからのオフセット
128	66	f	+0
129	69	i	+1
130	72	r	+2
131	73	s	+3
132	74	t	+4
133	例では未使用		+5
134	例では未使用		+6
135	例では未使用		+7
136	例では未使用		+0
137	例では未使用		+1
138	73	s	+2
139	65	e	+3
140	63	c	+4
141	6F	o	+5
142	6E	n	+6
143	64	d	+7

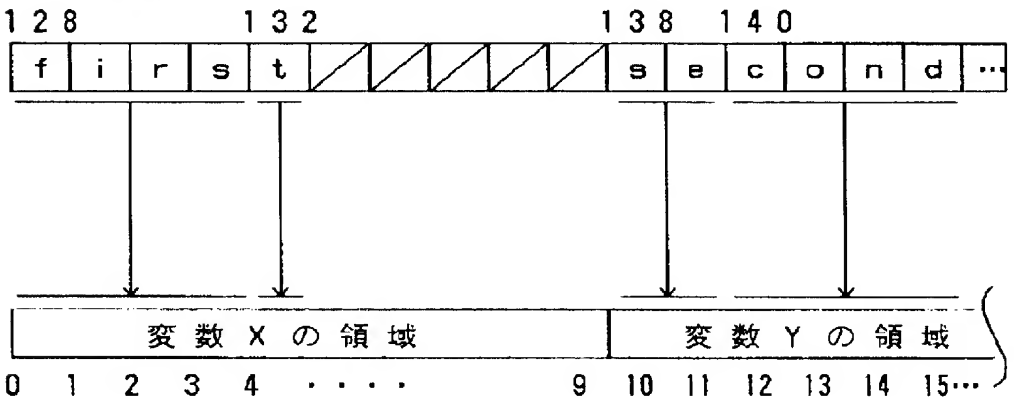
【図４】

命令の生成例説明図

(イ)

```
load_address      x,   r0      ;  変数Xのアドレスロード
load_4bytes       128, r1      ;  "firs"のコピー
store_4bytes      r1,  [r0]
load_1byte        132, r1      ;  "t" のコピー
store_1byte       r1,  [r0+4]
load_address      y,   r0      ;  変数Yのアドレスロード
load _2bytes      138, r1      ;  "se"のコピー
store_2bytes      r1,  [r0]
load_4bytes       140, r1      ;  "cond"のコピー
store_4bytes      r1,  [r0+2]
```

(ロ)



【図５】

従来の定数領域割り付けの例

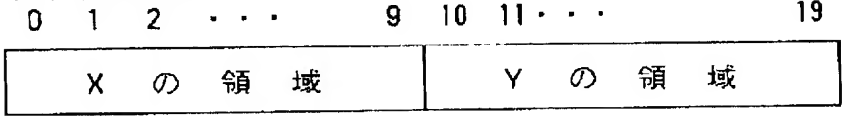
(イ)　ソースプログラム

```
01  TEST.
    02  X  PIC  X(10).
    02  Y  PIC  X(10).

...
MOVE  "FIRST"  TO  X.
MOVE  "SECOND"  TO  Y.
```

10

(ロ) 変数領域



(ハ) 定数領域

定数領域アドレス	16進表現	ASCII表現	8バイトアライメントからのオフセット
128	66	f	+0
129	69	i	+1
130	72	r	+2
131	73	s	+3
132	74	t	+4
133	73	s	+5
134	65	e	+6
135	63	c	+7
136	6F	o	+0
137	6E	n	+1
138	64	d	+2

【図６】

従来の命令の生成例説明図

(イ)

```
load_address    x,   r0      ;  変数Xのアドレスロード
load_4bytes     128, r1      ;  "firs"のコピー
store_4bytes    r1,  [r0]    ;
load_1byte      132, r1      ;  "t" のコピー
store_1byte     r1,  [r0+4]
load_address    y,   r0      ;  変数Yのアドレスロード
load_1byte      133, r1      ;  "s" のコピー
store_1byte     r1,  [r0]
load_1byte      134, r1      ;  "e" のコピー
store_1byte     r1,  [r0+1]
load_1byte      135, r1      ;  "c" のコピー
store_1byte     r1,  [r0+2]
load_1byte      136, r1      ;  "o" のコピー
store_1byte     r1,  [r0+3]
load_1byte      137, r1      ;  "n" のコピー
store_1byte     r1,  [r0+4]
load_1byte      138, r1      ;  "d" のコピー
store_1byte     r1,  [r0+5]
```

(ロ)